

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日
Date of Application: 2 0 0 2 年 9 月 2 4 日

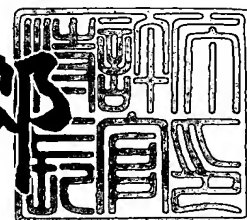
出 願 番 号
Application Number: 特 願 2 0 0 2 - 2 7 6 5 3 2
[ST. 10/C]: [J P 2 0 0 2 - 2 7 6 5 3 2]

出 願 人
Applicant(s): 株式会社リコー

2 0 0 3 年 7 月 9 日

特許庁長官
Commissioner,
Japan Patent Office

太田信一郎



出証番号 出証特 2 0 0 3 - 3 0 5 4 8 2 4

【書類名】 特許願

【整理番号】 0204683

【提出日】 平成14年 9月24日

【あて先】 特許庁長官殿

【国際特許分類】 G03G 21/00 370

【発明の名称】 画像形成装置およびアプリケーション生成方法

【請求項の数】 11

【発明者】

 【住所又は居所】 東京都大田区中馬込 1 丁目 3 番 6 号 株式会社リコー内

 【氏名】 秋吉 邦洋

【特許出願人】

 【識別番号】 000006747

 【氏名又は名称】 株式会社リコー

【代理人】

 【識別番号】 100089118

 【弁理士】

 【氏名又は名称】 酒井 宏明

【手数料の表示】

 【予納台帳番号】 036711

 【納付金額】 21,000円

【提出物件の目録】

 【物件名】 明細書 1

 【物件名】 図面 1

 【物件名】 要約書 1

 【包括委任状番号】 9808514

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 画像形成装置およびアプリケーション生成方法

【特許請求の範囲】

【請求項 1】 画像形成処理にかかるユーザサービスを提供するアプリケーションを備えた画像形成装置であって、

前記アプリケーションの取り得る状態と、前記状態下で前記アプリケーションに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の前記状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを含む前記アプリケーションに実装され、各モジュールで共有する共用変数を前記モジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリ

を備えたことを特徴とする画像形成装置。

【請求項 2】 前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記共用変数を、前記モジュールごとのスタック領域で指定された領域に割り当てることを特徴とする請求項 1 に記載の画像形成装置。

【請求項 3】 前記ユーティリティライブラリの前記状態遷移モデル起動関数は、スレッド単位で共用される共用変数を前記スレッドごとに別個の領域に割り当てることを特徴とする請求項 1 または 2 に記載の画像形成装置。

【請求項 4】 前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態の遷移が生じたときに行う処理を定めた入場関数を実行することを特徴とする請求項 1 ～ 3 のいずれか一つに記載の画像形成装置。

【請求項 5】 前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態とは無関係に発生する非同期イベントを受信したときに行う処理を定めた標準関数を実行することを特徴とする請求項 1 ～ 4 のいずれか一つに記載の画像形成装置。

【請求項 6】 画像形成処理で使用するハードウェア資源と、
前記アプリケーションと前記ハードウェア資源との間に介在し、ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前

記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスと、をさらに備え、

前記アプリケーションは、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うものであり、

前記ユーティリティライブラリは、前記アプリケーションに結合されていることを特徴とする請求項 1～5 のいずれか一つに記載の画像形成装置。

【請求項 7】 画像形成処理にかかるユーザサービスを提供する画像形成装置で動作するアプリケーションを生成するアプリケーション生成方法において、

前記アプリケーションの取り得る状態と、前記状態下で前記アプリケーションに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の前記状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを含む前記アプリケーションのオブジェクトファイルと、各モジュールで共有する共用変数を前記モジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリとを結合するリンクステップ

を含むことを特徴とするアプリケーション生成方法。

【請求項 8】 前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記共用変数を、前記モジュールごとのスタック領域で指定された領域に割り当てることを特徴とする請求項 7 に記載のアプリケーション生成方法。

【請求項 9】 前記ユーティリティライブラリの前記状態遷移モデル起動関数は、スレッド単位で共用される共用変数を前記スレッドごとに別個の領域に割り当てることを特徴とする請求項 7 または 8 に記載のアプリケーション生成方法。

【請求項 10】 前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態の遷移が生じたときに行う処理を定めた入場関数を実行することを特徴とする請求項 7～9 のいずれか一つに記載のアプリケーション生成方法。

【請求項 11】 前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態とは無関係に発生する非同期イベントを受信したときに行う処

理を定めた標準関数を実行することを特徴とする請求項7～10のいずれか一つに記載のアプリケーション生成方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

この発明は、コピー、プリンタ、スキャナおよびファクシミリなどの画像形成処理にかかるユーザサービスを提供し、状態遷移モデルに基づいてアプリケーションを開発するための関数が登録されたユーティリティライブラリを備えた画像形成装置およびアプリケーション生成方法に関するものである。

【0002】

【従来の技術】

近年では、プリンタ、コピー、ファクシミリ、スキャナなどの各装置の機能を1つの筐体内に収納した画像形成装置（以下、「複合機」という。）が知られている。この複合機は、1つの筐体内に表示部、印刷部および撮像部などを設けるとともに、プリンタ、コピーおよびファクシミリ装置にそれぞれ対応した3種類のソフトウェア（アプリケーション）を設け、これらのソフトウェアを切り替えることによって、当該装置をプリンタ、コピー、スキャナまたはファクシミリ装置として動作させるものである。

【0003】

このような従来の複合機では、複合機上で動作するアプリケーションの開発は、複合機の開発元で一括して行い、開発されたアプリケーションをROMなどに焼き付けた状態で複合機を出荷し、出荷後は新たなアプリケーションの追加を行わないのが一般的である。

【0004】

ところで、このような従来の複合機では、プリンタ、コピー、スキャナおよびファクシミリ装置に対応するソフトウェアをそれぞれ別個に設けているため、各ソフトウェアの開発に多大の時間を要する。このため、出願人は、表示部、印刷部および撮像部などの画像形成処理で使用されるハードウェア資源を有し、プリンタ、コピーまたはファクシミリなどの各ユーザサービスにそれぞれ固有の処理

を行うアプリケーションを複数搭載し、これらのアプリケーションとハードウェア資源との間に介在して、ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とするハードウェア資源の管理、実行制御並びに画像形成処理を行う各種コントロールサービスからなるプラットフォームを含む画像形成装置（複合機）を発明した。

【0005】

【発明が解決しようとする課題】

このような新規な複合機は、アプリケーションの少なくとも2つが共通的に必要とするハードウェア資源の管理、実行制御並びに画像形成処理を行うプラットフォームを備えた構成とすることによって、ソフトウェア開発の効率化を図るとともに、装置全体としての生産性を向上させることができる点で優れている。

【0006】

一方、このような新規な複合機では、アプリケーションと、ハードウェア資源にアクセスするような開発が難しい処理を行うコントロールサービスとが別個に設けられているため、複合機の出荷後にユーザもしくは第三者であるサードベンダが新規な外部アプリケーションを開発して複合機に搭載することができる構成となっている。

【0007】

このため、このような複合機では、複数のサードベンダによってそれぞれ別個に開発されたアプリケーションが複数搭載されて同時に動作する場合がある。また、一つのサードベンダが開発した一つのアプリケーションであっても、そのアプリケーション内部の複数のモジュールを別々の部署で自由に開発して一つのアプリケーションとして統合して提供する場合も考えられる。

【0008】

このように、アプリケーションの開発が複合機の開発メーカ以外のサードベンダ等に委ねた場合、各サードベンダが独自の手法でアプリケーション開発を行うこととなってしまうため、開発されたアプリケーション内部で障害が発生したり、出荷時に予め搭載されている既存のアプリケーションや複合機のシステムに悪影響を与えてしまうおそれがある。特に、アプリケーションプログラムで使用さ

れる共用変数や複合機独自のイベントについて全くその使用方法をサードベンダに委ねてしまうと、ソフトウェア障害によって複合機のシステムが不安定になってしまうおそれが高い。

【0009】

この発明は上記に鑑みてなされたもので、システムの安定性を保持しつつアプリケーションの追加を容易に行うことができる画像形成装置およびアプリケーション生成方法を得ることを目的とする。

【0010】

【課題を解決するための手段】

上記目的を達成するため、請求項1にかかる発明は、画像形成処理にかかるユーザサービスを提供するアプリケーションを備えた画像形成装置であって、前記アプリケーションの取り得る状態と、前記状態下で前記アプリケーションに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の前記状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを含む前記アプリケーションに実装され、各モジュールで共有する共用変数を前記モジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリを備えたことを特徴とする。

【0011】

この請求項1の発明によれば、アプリケーションの取り得る状態と、前記状態下で前記アプリケーションに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の前記状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを含む前記アプリケーションに実装され、各モジュールで共有する共用変数を前記モジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリを備えているので、アプリケーションを構成するモジュール間で共用変数を安全に使用することができ、変数の値に起因する障害を回避でき、画像形成装置に新規アプリケーションを搭載可能としながらも画像形成装置の安定性を維持することができる。

【0012】

また、請求項2にかかる発明は、請求項1に記載の画像形成装置において、前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記共用変数を、前記モジュールごとのスタック領域で指定された領域に割り当てることを特徴とする。

【0013】

この請求項2の発明によれば、状態遷移モデル起動関数は、前記共用変数を、前記モジュールごとのスタック領域で指定された領域に割り当てることで、関数呼び出しの際に使用されるスタック領域を利用してモジュールが呼び出しの際に効率的に共用変数の割り当てを行うことができる。

【0014】

また、請求項3にかかる発明は、請求項1または2に記載の画像形成装置において、前記ユーティリティライブラリの前記状態遷移モデル起動関数は、スレッド単位で共用される共用変数を前記スレッドごとに別個の領域に割り当てることを特徴とする。

【0015】

この請求項3の発明によれば、状態遷移モデル起動関数は、スレッド単位で共用される共用変数を前記スレッドごとに別個の領域に割り当てることで、アプリケーションのプロセス内部で動作するスレッド内部で共用変数を安全に使用することができ、変数の値に起因する障害を回避でき、画像形成装置に新規アプリケーションを搭載可能としながらも画像形成装置の安定性を維持することができる。

【0016】

また、請求項4にかかる発明は、請求項1～3のいずれか一つに記載の画像形成装置において、前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態の遷移が生じたときに行う処理を定めた入場関数を実行することを特徴とする。

【0017】

この請求項4の発明によれば、状態遷移モデル起動関数は、前記状態の遷移が

生じたときに行う処理を定めた入場関数を実行することで、モジュール呼び出しの際に定型的に行われる処理を実行することができ、アプリケーション開発を効率的に行える。

【0018】

また、請求項5にかかる発明は、請求項1～4のいずれか一つに記載の画像形成装置において、前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態とは無関係に発生する非同期イベントを受信したときに行う処理を定めた標準関数を実行することを特徴とする。

【0019】

この請求項5の発明によれば、状態遷移モデル起動関数は、前記状態とは無関係に発生する非同期イベントを受信したときに行う処理を定めた標準関数を実行することで、現在の状態とは無関係に生じるシステム関連のイベントなどに他のアプリケーションに悪影響を与えずに対処することができ、画像形成装置の安定性を向上させることができる。

【0020】

また、請求項6にかかる発明は、請求項1～5のいずれか一つに記載の画像形成装置において、画像形成処理で使用されるハードウェア資源と、前記アプリケーションと前記ハードウェア資源との間に介在し、ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスと、をさらに備え、前記アプリケーションは、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うものであり、前記ユーティリティライブラリは、前記アプリケーションに結合されていることを特徴とする。

【0021】

この請求項6の発明によれば、画像形成処理で使用されるハードウェア資源と、前記アプリケーションと前記ハードウェア資源との間に介在し、ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスと、をさらに備え、前記アプリケーションは、画像形成処理に

かかるユーザサービスにそれぞれ固有の処理を行うものであり、前記ユーティリティライブラリは、前記アプリケーションに結合されているので、アプリケーションのソフトウェア開発をユーザサービスに固有の部分だけで行うことができ、ソフトウェア開発の労力軽減を図ることができる。

【0022】

また、請求項7にかかる発明は、画像形成処理にかかるユーザサービスを提供する画像形成装置で動作するアプリケーションを生成するアプリケーション生成方法において、前記アプリケーションの取り得る状態と、前記状態下で前記アプリケーションに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の前記状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを含む前記アプリケーションのオブジェクトファイルと、各モジュールで共有する共用変数を前記モジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリとを結合するリンクステップを含むことを特徴とする。

【0023】

この請求項7の発明によれば、アプリケーションの取り得る状態と、前記状態下で前記アプリケーションに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の前記状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを含む前記アプリケーションのオブジェクトファイルと、各モジュールで共有する共用変数を前記モジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリとを結合するリンクステップを含むことで、アプリケーションを構成するモジュール間で共用変数を安全に使用することができ、変数の値に起因する障害を回避でき、画像形成装置に新規アプリケーションを搭載可能としながらも画像形成装置の安定性を維持することができる。

【0024】

また、請求項8にかかる発明は、請求項7に記載のアプリケーション生成方法

において、前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記共用変数を、前記モジュールごとのスタック領域で指定された領域に割り当てることを特徴とする。

【0025】

この請求項8の発明によれば、状態遷移モデル起動関数は、前記共用変数を、前記モジュールごとのスタック領域で指定された領域に割り当てることで、関数呼び出しの際に使用されるスタック領域を利用してモジュールが呼び出しの際に効率的に共用変数の割り当てを行うことができる。

【0026】

また、請求項9にかかる発明は、請求項7または8に記載のアプリケーション生成方法において、前記ユーティリティライブラリの前記状態遷移モデル起動関数は、スレッド単位で共用される共用変数を前記スレッドごとに別個の領域に割り当てることを特徴とする。

【0027】

この請求項9の発明によれば、状態遷移モデル起動関数は、スレッド単位で共用される共用変数を前記スレッドごとに別個の領域に割り当てることで、アプリケーションのプロセス内部で動作するスレッド内部で共用変数を安全に使用することができ、変数の値に起因する障害を回避でき、画像形成装置に新規アプリケーションを搭載可能としながらも画像形成装置の安定性を維持することができる。

【0028】

また、請求項10にかかる発明は、請求項7～9のいずれか一つに記載のアプリケーション生成方法において、前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態の遷移が生じたときに行う処理を定めた入場関数を実行することを特徴とする。

【0029】

この請求項10の発明によれば、状態遷移モデル起動関数は、前記状態の遷移が生じたときに行う処理を定めた入場関数を実行することで、モジュール呼び出しの際に定型的に行われる処理を実行することができ、アプリケーション開発を

効率的に行える。

【0030】

また、請求項 11 にかかる発明は、請求項 7～10 のいずれか一つに記載のアプリケーション生成方法において、前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態とは無関係に発生する非同期イベントを受信したときに行う処理を定めた標準関数を実行することを特徴とする。

【0031】

この請求項 11 の発明によれば、状態遷移モデル起動関数は、前記状態とは無関係に発生する非同期イベントを受信したときに行う処理を定めた標準関数を実行することで、現在の状態とは無関係に生じるシステム関連のイベントなどに他のアプリケーションに悪影響を与えずに対処することができ、画像形成装置の安定性を向上させることができる。

【0032】

【発明の実施の形態】

以下に添付図面を参照して、この発明にかかる画像形成装置およびアプリケーション生成方法の好適な実施の形態を詳細に説明する。

【0033】

(実施の形態 1)

図 1 は、この発明の実施の形態 1 である画像形成装置（以下、「複合機」という。）の構成を示すブロック図である。

【0034】

図 1 に示すように、複合機 100 は、白黒レーザプリンタ (B&W LP) 101 と、カラーレーザプリンタ (Color LP) 102 とハードディスク装置 (HDD) 103 と、ファクシミリ、メモリなどのハードウェアリソース 104 を有するとともに、プラットフォーム 120 と、アプリケーション 130 とから構成されるソフトウェア群 110 とを備えている。

【0035】

プラットフォーム 120 は、アプリケーションからの処理要求を解釈してハードウェア資源の獲得要求を発生させるコントロールサービスと、一または複数のハ

ードウェア資源の管理を行い、コントロールサービスからの獲得要求を調停するシステムリソースマネージャ（SRM）123と、汎用OS121とを有する。

【0036】

コントロールサービスは、複数のサービスモジュールから形成され、SCS（システムコントロールサービス）122と、ECS（エンジンコントロールサービス）124と、MCS（メモリコントロールサービス）125と、OCS（オペレーションパネルコントロールサービス）126と、FCS（ファックスコントロールサービス）127と、NCS（ネットワークコントロールサービス）128とから構成される。このプラットフォーム120は、あらかじめ定義された関数により前記アプリケーション130から処理要求を受信可能とするアプリケーションプログラムインタフェース（API）を有する。

【0037】

汎用OS121は、UNIX（登録商標）などの汎用オペレーティングシステムであり、プラットフォーム120並びにアプリケーション130の各ソフトウェアをそれぞれプロセスとして並列実行する。

【0038】

SRM123のプロセスは、SCS122とともにシステムの制御およびリソースの管理を行うものである。SRM123のプロセスは、スキャナ部やプリンタ部などのエンジン、メモリ、HDDファイル、ホストI/O（セントロI/F、ネットワークI/F、IEEE1394 I/F、RS232C I/Fなど）のハードウェア資源を利用する上位層からの要求にしたがって調停を行い、実行制御する。

【0039】

具体的には、このSRM123は、要求されたハードウェア資源が利用可能であるか（他の要求により利用されていないかどうか）を判断し、利用可能であれば要求されたハードウェア資源が利用可能である旨を上位層に伝える。また、SRM123は、上位層からの要求に対してハードウェア資源の利用スケジューリングを行い、要求内容（例えば、プリンタエンジンにより紙搬送と作像動作、メモリ確保、ファイル生成など）を直接実施している。

【0040】

SCS122のプロセスは、アプリ管理、操作部制御、システム画面表示、LED表示、リソース管理、割り込みアプリ制御などを行う。

【0041】

ECS124のプロセスは、白黒レーザプリンタ (B&W LP) 101、カラーレーザプリンタ (Color LP) 102、HDD103、スキャナ、ファクシミリなどからなるハードウェアリソース104のエンジンの制御を行う。

【0042】

MCS125のプロセスは、画像メモリの取得および解放、ハードディスク装置 (HDD) の利用、画像データの圧縮および伸張などを行う。

【0043】

FCS127のプロセスは、システムコントローラの各アプリ層からPSTN／ISDN網を利用したファクシミリ送受信、BKM (バックアップSRAM) で管理されている各種ファクシミリデータの登録／引用、ファクシミリ読みとり、ファクシミリ受信印刷、融合送受信を行うためのAPIを提供する。

【0044】

NCS128のプロセスは、ネットワークI/Oを必要とするアプリケーションに対して共通に利用できるサービスを提供するためのプロセスであり、ネットワーク側から各プロトコルによって受信したデータを各アプリケーションに振り分けたり、アプリケーションからデータをネットワーク側に送信する際の仲介を行う。

【0045】

OCS126のプロセスは、オペレータ (ユーザ) と本体制御間の情報伝達手段となるオペレーションパネル (操作パネル) の制御を行う。OCS126は、オペレーションパネルからキー押下 (またはタッチ操作) をキーイベントとして取得し、取得したキーに対応したキーイベント関数をSCS122に送信するOCSプロセスである。また、オペレーションパネルの操作表示部に対する各種画面を描画出力やその他オペレーションパネルに対する制御は、OCS関数ライブラリに登録されている描画関数等の各種関数をアプリケーション130またはコ

ントロールサービスから呼び出すことにより行われる。このOCS関数ライブラリは、アプリケーション130およびコントロールサービスの各モジュールに動的にリンクされている。なお、OCS126のすべてをプロセスとして動作させるように構成しても良く、あるいはOCS126のすべてをOCS関数ライブラリとして構成しても良い。

【0046】

アプリケーション130は、ページ記述言語(PDL)、PCLおよびポストスクリプト(PS)を有するプリンタ用のアプリケーションであるプリンタアプリ111と、コピー用アプリケーションであるコピーアプリ112と、ファクシミリ用アプリケーションであるファックスアプリ113と、スキャナ用アプリケーションであるスキャナアプリ114と、ネットワークファイル用アプリケーションであるネットファイルアプリ115と、工程検査用アプリケーションである工程検査アプリ116と、サードベンダなどの第三者が開発した外部アプリ117とを有している。外部アプリ117は、状態遷移モデルに従った開発手法により生成されており、状態遷移モデル起動関数が登録されたユーティリティライブラリをリンクしている。

【0047】

アプリケーション130の各プロセス、コントロールサービスの各プロセスは、関数呼び出しとその戻り値送信およびメッセージの送受信によってプロセス間通信を行いながら、コピー、プリンタ、スキャナ、ファクシミリなどの画像形成処理にかかるユーザサービスを実現している。

【0048】

このように、実施の形態1にかかる複合機100には、複数のアプリケーション130および複数のコントロールサービスが存在し、いずれもプロセスとして動作している。そして、これらの各プロセス内部には、一または複数のスレッドが生成されて、スレッド単位の並列実行が行われる。そして、コントロールサービスがアプリケーション130に対し共通サービスを提供しており、このため、これらの多数のプロセスが並列動作、およびスレッドの並列動作を行って互いにプロセス間通信を行って協調動作をしながら、コピー、プリンタ、スキャナ、フ

ァクシミリなどの画像形成処理にかかるユーザサービスを提供するようになっている。

【0049】

また、複合機100には、複合機100の顧客、サードベンダなどの第三者がコントロールサービス層の上のアプリケーション層に外部アプリを開発して搭載することが可能となっている。

【0050】

なお、実施の形態1にかかる複合機100では、複数のアプリケーション130のプロセスと複数のコントロールサービスのプロセスとが動作しているが、アプリケーション130とコントロールサービスのプロセスがそれぞれ単一の構成とすることも可能である。また、各アプリケーション130は、アプリケーションごとに追加または削除することができる。

【0051】

次に、外部アプリ117が実行されている場合のプロセス内部の構造について説明する。図2は、外部アプリ117のプロセス内部の構造を示す説明図である。

【0052】

外部アプリ117のプロセスは、複数のスレッドが並列実行されたマルチスレッド環境となっている。図2に示すように、外部アプリ117のプロセスには、メインスレッド210とイメージライブラリスレッド200が並列実行されている。メインスレッド210は、外部アプリ117の主要な処理を実行するスレッドであり、このメインスレッド210がイメージライブラリスレッド200を呼び出すことにより、コントロールサービス140やVAS（仮想アプリケーションサービス）などの関数呼び出しとその戻り値の受信、メッセージの送受信を実現している。

【0053】

また、外部アプリ117のプロセスには、サブスレッド220、メールボックス211、212、214、イベント取得ハンドラ（スレッド）213が存在している。メインスレッド210には、状態遷移モデル起動関数が登録されたユー

ティリティライブラリ 2 1 5 が実装されている。

【0 0 5 4】

メールボックス 2 1 1 は、メインスレッド 2 1 0 とイメージライブラリスレッド 2 0 0 およびイベント取得ハンドラ 2 1 3 との通信に使用される。メールボックス 2 1 2 は、メインスレッド 2 1 0 とイメージライブラリスレッド 2 0 0 の通信に使用される。イベント取得ハンドラ 2 1 3 は、E C S 1 2 4, M C S 1 2 5 などのコントロールサービスまたは V A S のプロセスからのイベントメッセージを受信し、メールボックス 2 1 1 に書き込むものである。

【0 0 5 5】

メインスレッド 2 1 0 がコントロールサービスなどにアクセスするためには、メインスレッド 3 1 0 がメールボックス 2 1 1, 2 1 2, 2 1 4 を開設した後、イメージライブラリスレッドを起動する。

【0 0 5 6】

メインスレッド 3 1 0 は、イメージライブラリスレッド 2 0 0 に対する関数呼び出しのコマンドを、送信元情報が M B _ R E Q U E S T としてメールボックス 2 1 1 に入れる。また、コントロールサービス 1 4 0 などからのイベントメッセージも、送信元情報 M B _ C S のメールとしてメールボックス 2 1 1 に入るようになっている。

【0 0 5 7】

イメージライブラリスレッド 2 0 0 は、メールボックス 2 1 1 に配信されるメールを `r c v _ m s g ()` 関数によって取得する。イメージライブラリスレッド 2 0 0 は、取得したメールに含まれる送信元情報とイベントまたは関数情報から、対応する関数の検索を行い、対応する関数が検索された場合にその関数を実行する。一方、対応する関数が検索されなかった場合には何も処理は行わない。

【0 0 5 8】

取得したメールは対応する関数の実行に関係なく、メールボックス 2 1 2 に転送される。また、イメージライブラリスレッド 2 0 0 は、メインスレッドに対して、エラー情報、実行結果、途中経過などの通知を送信元情報 M B _ I M A G E のメールとして、メールボックス 2 1 2 に入れる。また、メインスレッド 2 1 0

は、サブスレッド220とメールボックス214、212を利用して同様にメッセージの送受信を行う。

【0059】

また、ユーティリティライブラリ215およびイメージライブラリは、複合機のアプリケーション開発のためのソフトウェア開発キット（SDK: Software Development Kit）などの開発用ツールキットの一部または全部として、CD-ROMまたはFDなどの記憶媒体にインストール可能な形式のファイルで提供される。また、ユーティリティライブラリ215およびイメージライブラリを、Webサイト等からのダウンロード等、ネットワーク経由で取得可能な方法で提供するようにしても良い。

【0060】

このように提供された上記ユーティリティライブラリ215およびイメージライブラリを用いて外部アプリ117（複合機100の出荷後に第三者によって開発されて搭載されるアプリ）を生成するには、外部アプリのソースファイルをコンパイラによってコンパイルしてオブジェクトファイルを生成する。そして、リンカによって、生成されたオブジェクトファイルと、上記の方法で提供されたユーティリティライブラリ215およびイメージライブラリとをリンクして、外部アプリの実行形式ファイルを生成する。この外部アプリの実行形式ファイルをフラッシュカードなどに格納して、複合機100において、フラッシュカードから外部アプリをインストールすることにより、ユーティリティライブラリ215およびイメージライブラリを実装した外部アプリが複合機に搭載される。

【0061】

次に、外部アプリ117が採用している状態遷移モデルについて説明する。図3は、外部アプリ117が状態1と状態2を有する場合の状態遷移の説明図である。図3に示すように、外部アプリ117のメインスレッド210が、状態1および状態2を有し、現在の状態が状態1であると仮定する。このメインスレッド210が状態1にある状態で外部からrcv_msg()関数によってイベント1を受信した場合、イベント関数1を実行して状態2に遷移し、入場関数2を実行する。メインスレッド210がこの状態2にあるときにイベント2を受信する

と、イベント関数2を実行して状態1に戻り、入場関数1を実行する。一方、メインスレッド210が状態2にあるときにイベント3を受信すると、イベント関数3を実行し、状態2を維持する。

【0062】

ここで、状態とは、イベントを待機する状態をいい、イベントとは、状態遷移のトリガとなる事象である。また、入場関数とは、状態遷移の際に実行される関数であり、イベント関数とは、発生したイベントに対して実行される関数である。

【0063】

このような状態遷移モデルを利用してイベント処理を行う外部アプリ117を作成する際に、発生する全てのイベントを単一のモジュール、すなわち同じレベルの状態群で対応しようとする、状態数の増大、一つの状態で処理するイベント数の増大が発生し、プログラムが複雑になり作業効率の低下、メンテナンス効率の低下という問題が生じる。

【0064】

このため、本実施の形態では、外部アプリ117の全体で発生するイベント群から相互関係のあるイベントのみを処理する状態遷移モデルのモジュール（サブモジュール）を個別に作成している。そして、全体を制御する状態遷移モデルのモジュールからは、必要に応じて上記サブモジュールを関数呼び出しの形式で実行する。呼び出しを受けたサブモジュールは、必要に応じてさらに状態遷移モデルのサブモジュールの関数呼び出しを行う。

【0065】

各モジュールは、外部アプリ117の全体で利用できるグローバル変数を使用せずに、起動時に共用するメモリを動的に確保することによって、同じモジュールが再帰的に呼び出されても正確に実行することが保証される。

【0066】

図4は、ネスト構造を有する状態遷移モデルのモジュールからなる外部アプリ117の一例を示す説明図である。図4に示す例では、3つのネスト構造を有しており、イベント関数sub1()によってネスト1のモジュールが呼び出され

、イベント関数 `sub2()` によってネスト1のモジュールが呼び出されている。

【0067】

次に、外部アプリ117のソースコードの具体的な記述について説明する。状態遷移モデルのモジュールで実行する入場関数、イベント関数において関数間で共用した変数がある場合、関数型の制約上、その変数を関数の引数として渡すことができない。また、このような変数をグローバル変数として定義すると、ネスト構造のモジュールやマルチスレッドの場合、予期せぬ変数の内容の変更が行われる可能性が高く、外部アプリ117の障害原因となる。

【0068】

本実施の形態の外部アプリ117では、単一の状態遷移モデルのモジュールおよびネストしているモジュールで変数を共用可能としている。図5は、外部アプリ117のソースコードの定義部の一例を示す説明図である。

【0069】

図5に示すように、外部アプリ117のソースコードでは、同一ファイルに記述された入場関数、イベント関数でグローバル変数のように扱う変数を集めて、共用変数として一つの構造体 (`demoCommonParm_t`) を定義する。また、この共用変数に簡易にアクセスするために `COMMON_STRUCT` を定義している。

【0070】

また、図5の例では、外部アプリ117は、2つの状態 `DEMO_ST_IDLE`、`DEMO_ST_RUN` を有し、各状態のイベント関数テーブル `eventDemoIdle`、`eventDemoRun` が定義されている。また、`demoStTbl[]` は、この状態遷移モデルのモジュールで使用する各状態の入場関数、イベント関数テーブルを定義した状態の順に記述している。また、`THREAD_NO` によってスレッド番号を指定することにより、このスレッド番号のスレッド内で有効な共用変数を使用することが可能となる。

【0071】

図6は、外部アプリ117のソースコードの関数定義部の一例を示す説明図で

ある。execDemoFunc関数は、状態遷移モデルを起動する処理を行う。具体的には、共用変数の領域確保、設定の後、状態遷移モデル起動関数StMachine () 関数呼び出しを行う。図6の例では、スレッド番号THREAD_NOのスレッドで、demoStTbleの状態情報テーブルに従い、状態番号DEMO_ST_IDLEから起動し、pComが示す領域を共用変数として使用することを指示して、状態遷移モデル起動関数StMachine () を呼び出している。図6では、さらに図5で記述したイベント関数、入場関数の処理を記述する。

【0072】

次に、状態遷移モデル起動関数StMachine () の処理について説明する。図7は、状態遷移モデル起動関数StMachine () の処理手順を示すフローチャートである。

【0073】

状態遷移モデル起動関数StMachine () では、メッセージ受信を行うと、まず共用変数の領域割り当てを行う（ステップS701）。図8は、共用変数の領域割り当ての状態を示す説明図である。図8に示すように、本実施の形態ではスタックポインタspの値がそのままネスト番号に相当するようになっており、各ネストのモジュールごと、すなわち各ネストのモジュールで状態遷移モデル起動関数StMachine () が実行されるたびに、共用変数群へのポインタがスタック領域に積まれるようになっている。従って、共用変数が下位モジュールで不用意に変更されることがなく、図8に示すように同じ変数名の共用変数を正確な値に維持しつつグローバル変数のように使用することができる。

【0074】

次に、状態遷移モデル起動関数StMachine () は、現在の状態のイベント関数テーブルを参照してその内容をチェックする（ステップS702）。そして、イベント関数テーブルに受信メッセージが定義されているか否かを判断する（ステップS703）。そして、受信メッセージが定義されている場合には、受信メッセージに対応するイベント関数を実行し（ステップS704）、さらに入場関数を実行する（ステップS705）。一方、イベント関数テーブルに受信

メッセージが定義されていない場合には、イベント関数の実行は行わない。そして、このような処理をイベント関数テーブルの末尾に到達するまで繰り返し行う（ステップ S 7 0 6）。このようにして状態遷移モデルに基づいて作成された外部アプリ 1 1 7 の実行が行われる。

【0 0 7 5】

このように実施の形態 1 の複合機では、外部アプリ 1 1 7 が複数のネスト構造のモジュールを含む外部アプリ 1 1 7 に実装され、各モジュールで共有する共用変数をモジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリを 2 1 5 を有しているので、外部アプリ 1 1 7 を構成するモジュール間で共用変数を安全に使用することができ、変数の値に起因する障害を回避でき、複合機 1 0 0 に外部アプリ 1 1 7 を搭載可能としながらも複合機 1 0 0 の安定性を維持することができる。

【0 0 7 6】

（実施の形態 2）

実施の形態 1 の複合機 1 0 0 は、外部アプリ 1 1 7 が状態遷移モデルに基づいて作成されているため、状態に依存したイベント関数を実行するのみであったが、この実施の形態 2 の複合機 1 0 0 では、外部アプリ 1 1 7 が、さらに状態に依存しない標準処理を行うものである。なお、実施の形態 2 の複合機 1 0 0 および外部アプリ 1 1 7 のプロセス内部の構造は、実施の形態 1 と同様である。

【0 0 7 7】

実施の形態 2 の複合機 1 0 0 では、ユーティリティライブラリの状態遷移モデル起動関数 `StMachine()` が、標準処理を行うように構成されている。

【0 0 7 8】

図 9 は、状態遷移モデル起動関数 `StMachine()` の処理手順を示すフローチャートである。共用変数の割り当てから入場関数の実行までの処理（ステップ S 9 0 1 ～ S 9 0 6）は、実施の形態 1 の状態遷移モデル起動関数 `StMachine()` と同様である。

【0 0 7 9】

本実施の形態の状態遷移モデル起動関数 `StMachine()` では、さらに

標準関数テーブルを参照し、その内容をチェックする（ステップS907）。ここで、標準関数とは、現在の状態とはイベントを受信したときに無関係、非依存に実行される処理の関数である。例えば、電源OFFのイベントに対する処理、アプリケーションのオーナー権限の問い合わせイベントに対し、オーナー権限の有無を通知する処理などがあげられる。なお、標準関数テーブルは、メッセージ（イベント）に対して実行される標準関数を定めたものであり、イベント関数テーブルと同様の構造を有している。

【0080】

そして、標準関数テーブルに受信メッセージが定義されているか否かを判断する（ステップS908）。そして、受信メッセージが定義されている場合には、受信メッセージに対応する標準関数を実行する（ステップS909）。一方、標準関数テーブルに受信メッセージが定義されていない場合には、標準関数の実行は行わない。そして、このような処理を標準関数テーブルの末尾に到達するまで繰り返し行う（ステップS910）。このようにして状態遷移モデルに基づいて作成された外部アプリ117において、現在の状態に無関係に非同期にくるイベントに対する標準処理の実行が行われる。

【0081】

このように実施の形態2の複合機100では、状態遷移モデル起動関数は、現在の状態とは無関係に発生する非同期イベントを受信したときに行う処理を定めた標準関数を実行しているので、現在の状態とは無関係に生じる電源OFFなどのシステム関連のイベントなどに他のアプリに悪影響を与えずに対処することができ、複合機100の安定性を向上させることができる。

【0082】

【発明の効果】

以上説明したように、請求項1にかかる発明によれば、アプリケーションを構成するモジュール間で共用変数を安全に使用することができ、変数の値に起因する障害を回避でき、画像形成装置に新規アプリケーションを搭載可能としながらも画像形成装置の安定性を維持することができるという効果を奏する。

【0083】

また、請求項2にかかる発明によれば、関数呼び出しの際に使用されるスタック領域を利用してモジュールが呼び出しの際に効率的に共用変数の割り当てを行うことができるという効果を奏する。

【0084】

また、請求項3にかかる発明によれば、アプリケーションのプロセス内部で動作するスレッド内部で共用変数を安全に使用することができ、変数の値に起因する障害を回避でき、画像形成装置に新規アプリケーションを搭載可能としながらも画像形成装置の安定性を維持することができるという効果を奏する。

【0085】

また、請求項4にかかる発明によれば、モジュール呼び出しの際に定型的に行われる処理を実行することができ、アプリケーション開発を効率的に行えるという効果を奏する。

【0086】

また、請求項5にかかる発明によれば、現在の状態とは無関係に生じるシステム関連のイベントなどに他のアプリケーションに悪影響を与えずに対処することができ、画像形成装置の安定性を向上させることができるという効果を奏する。

【0087】

また、請求項6にかかる発明によれば、アプリケーションのソフトウェア開発をユーザサービスに固有の部分だけで行うことができ、ソフトウェア開発の労力軽減を図ることができるという効果を奏する。

【0088】

また、請求項7にかかる発明によれば、アプリケーションを構成するモジュール間で共用変数を安全に使用することができ、変数の値に起因する障害を回避でき、画像形成装置に新規アプリケーションを搭載可能としながらも画像形成装置の安定性を維持することができるという効果を奏する。

【0089】

また、請求項8にかかる発明によれば、関数呼び出しの際に使用されるスタック領域を利用してモジュールが呼び出しの際に効率的に共用変数の割り当てを行うことができるという効果を奏する。

【0090】

また、請求項9にかかる発明によれば、アプリケーションのプロセス内部で動作するスレッド内部で共用変数を安全に使用することができ、変数の値に起因する障害を回避でき、画像形成装置に新規アプリケーションを搭載可能としながらも画像形成装置の安定性を維持することができるという効果を奏する。

【0091】

また、請求項10にかかる発明によれば、モジュール呼び出しの際に定型的に行われる処理を実行することができ、アプリケーション開発を効率的に行えるという効果を奏する。

【0092】

また、請求項11にかかる発明によれば、現在の状態とは無関係に生じるシステム関連のイベントなどに他のアプリケーションに悪影響を与えずに対処することができ、画像形成装置の安定性を向上させることができるという効果を奏する。

【図面の簡単な説明】

【図1】

実施の形態1にかかる複合機の機能的構成を示すブロック図である。

【図2】

外部アプリのプロセス内部の構造を示す説明図である。

【図3】

外部アプリが状態1と状態2を有する場合の状態遷移の説明図である。

【図4】

ネスト構造を有する状態遷移モデルのモジュールからなる外部アプリの一例を示す説明図である。

【図5】

外部アプリのソースコードの定義部の一例を示す説明図である。

【図6】

外部アプリのソースコードの関数定義部の一例を示す説明図である。

【図7】

状態遷移モデル起動関数 `StMachine()` の処理手順を示すフローチャートである。

【図 8】

共用変数の領域割り当ての状態を示す説明図である。

【図 9】

実施の形態 2 における状態遷移モデル起動関数 `StMachine()` の処理手順を示すフローチャートである。

【符号の説明】

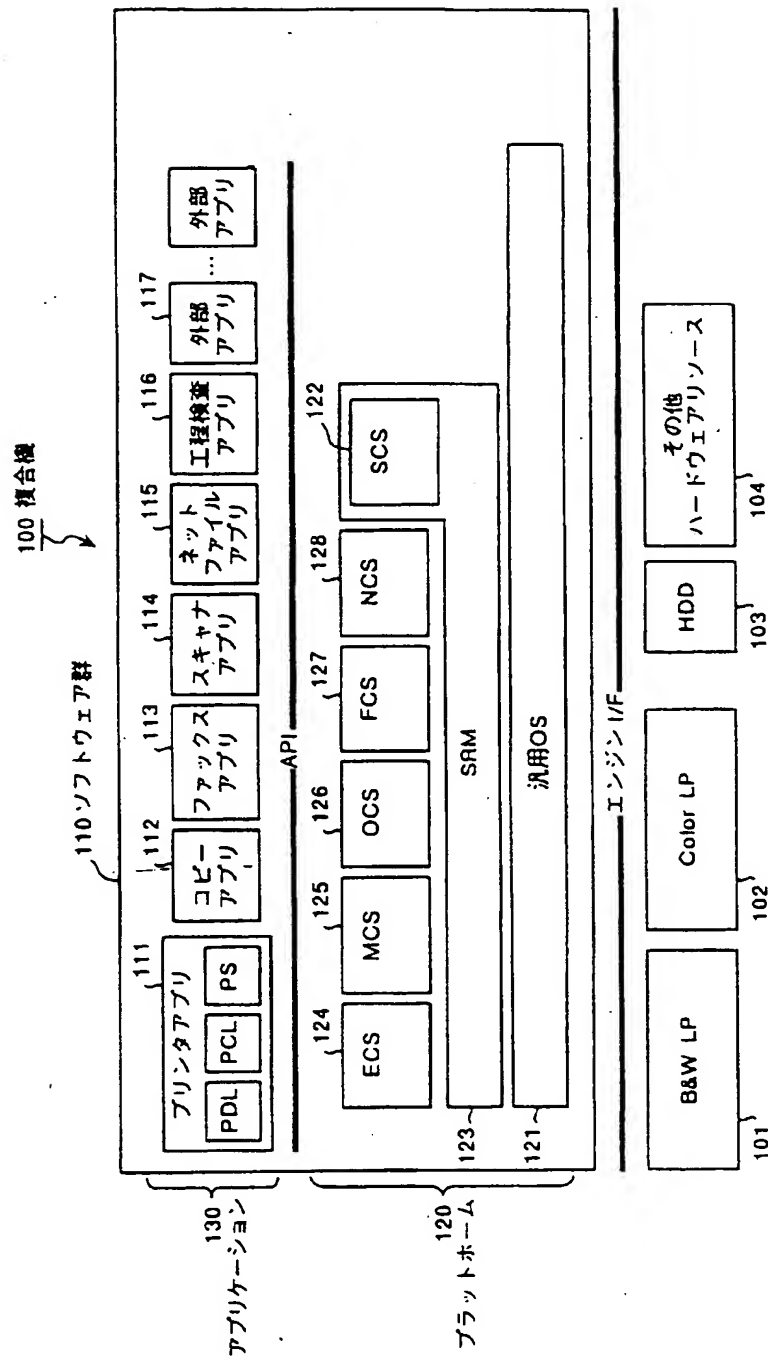
- 100 複合機
- 101 白黒レーザプリンタ
- 102 カラーレーザプリンタ
- 103 HDD
- 104 ハードウェアリソース
- 110 ソフトウェア群
- 111 プリンタアプリ
- 112 コピーアプリ
- 113 ファックスアプリ
- 114 スキャナアプリ
- 115 ネットファイルアプリ
- 116 工程検査アプリ
- 117 外部アプリ
- 120 プラットホーム
- 121 汎用 OS
- 122 SCS
- 123 SRM
- 124 ECS
- 125 MCS
- 126 OCS
- 127 FCS

1 2 8 N C S
1 3 0 アプリケーション
1 4 0 コントロールサービス
2 0 0 イメージライブラリスレッド
2 1 0 メインスレッド
2 1 1, 2 1 2, 2 1 4 メールボックス
2 1 3 イベント取得ハンドラ
2 1 5 ユーティリティライブラリ
2 2 0 サブスレッド

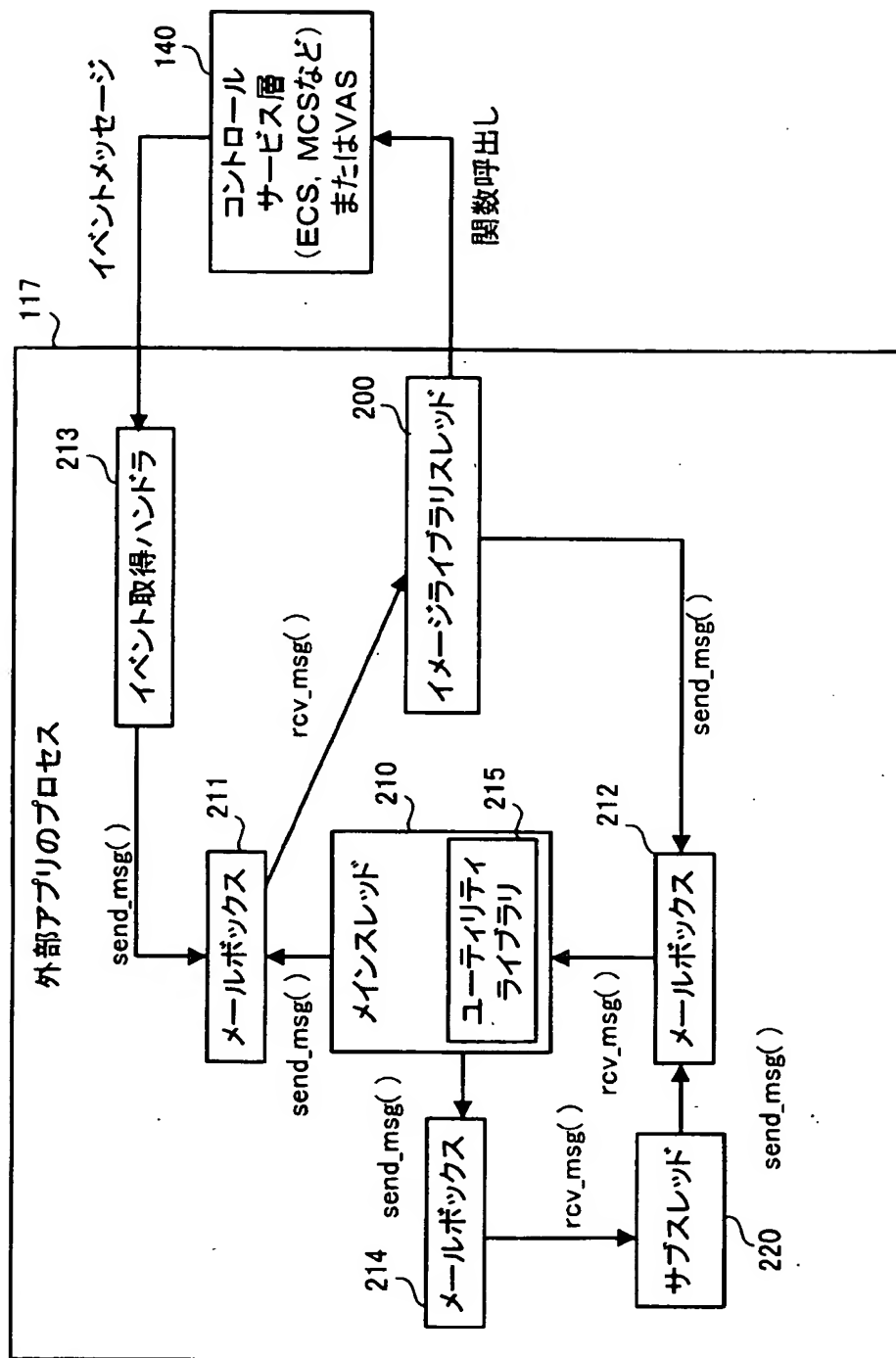
【書類名】

図面

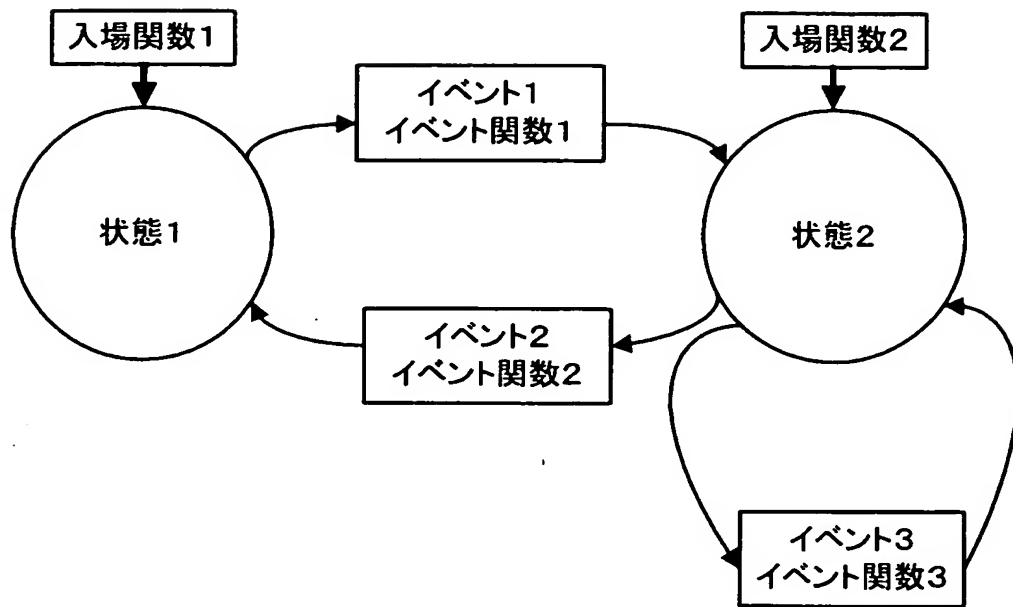
【図 1】



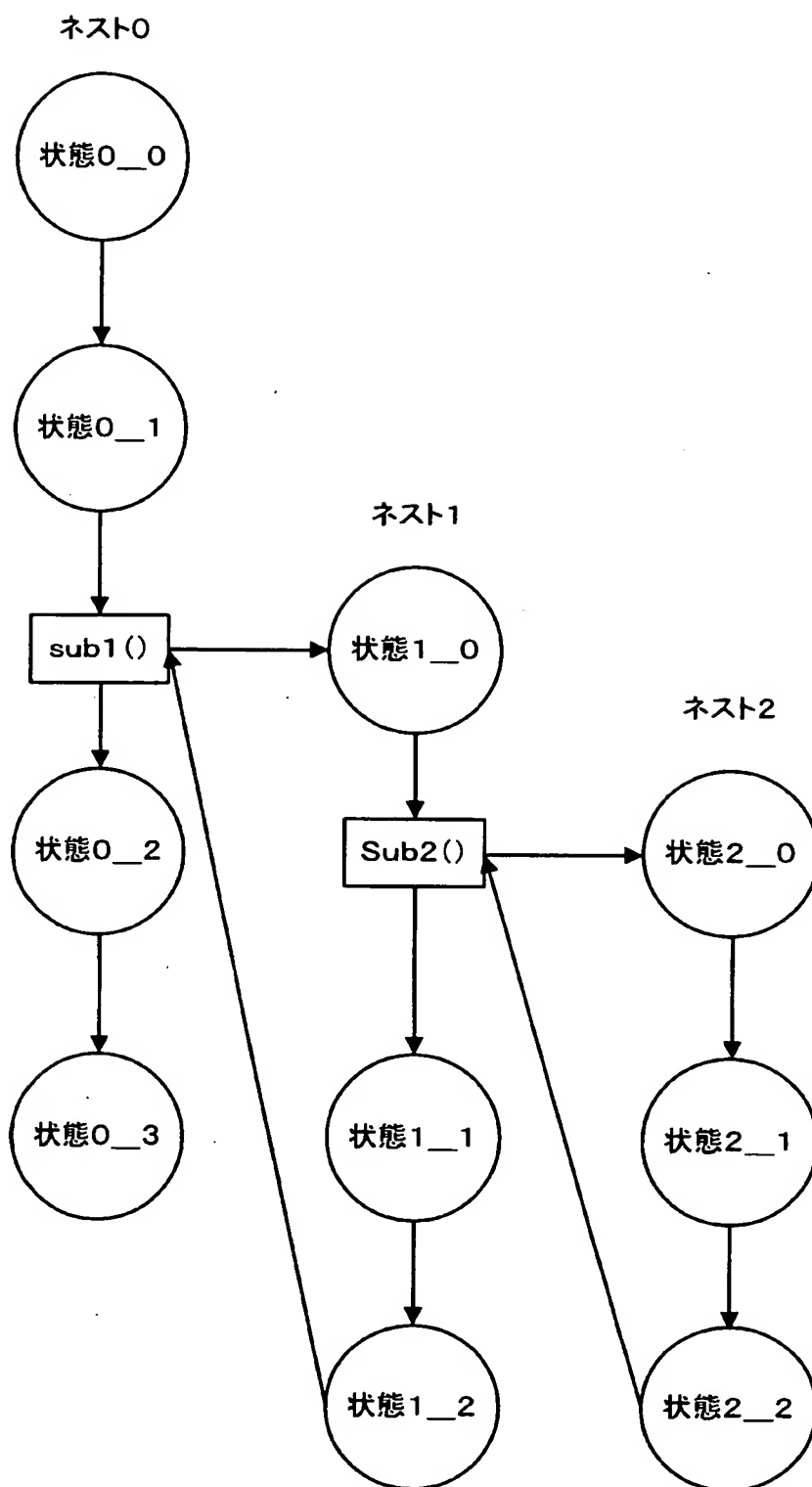
【図 2】



【図 3】



【図 4】



【図 5】

```
#define THREAD_NO 1          // スレッド番号:1
typedef struct {
    int      demoData;        // 共用変数の定義
} demoCommonParm_t;

#undef COMMON_STRUCT
#define COMMON_STRUCT demoCommonParm_t // 共用変数アクセスのための定義

enum DEMO_STATE{             // 状態の定義
    DEMO_ST_IDLE,
    DEMO_ST_RUN
};

owner TypeFunc_t eventDemoIdle[] = // 状態DEMO_ST_IDLEのイベント関数テーブル
{
    {      1,      evDemoIdle },
    {      0,      0           }
};

owner TypeFunc_t eventDemoRun[] = // 状態DEMO_ST_RUNのイベント関数テーブル
{
    {      2,      evDemoRun },
    {      0,      0           }
};

stateInfo_t demoStTbl[] = // 状態情報テーブル
{
    {      enterDemoIdle, eventDemoIdle },
    {      enterDemoRun,  eventDemoRun  }
};
```


【図 6】

```

/*****
    execDemoFunc
*****/
void    execDemoFunc(void)
{
    int ret;
    COMMON_STRUCT *pCom;    // 共用変数構造体ポインタの定義
    pCom = (COMMON_STRUCT *)malloc(sizeof(COMMON_STRUCT));
                                // 共用変数の領域確保
    pCom->demoData = 0;    // 共用変数の設定

    ret = StMachine(THREAD_NO, demoStTble, DEMO_ST_IDLE, pCom); // 起動
    free(pCom);            // 共用変数の開放
}

/*****
    enterDemoIdle  入場関数
*****/
void    enterDemoIdle(void)
{
    ...
}

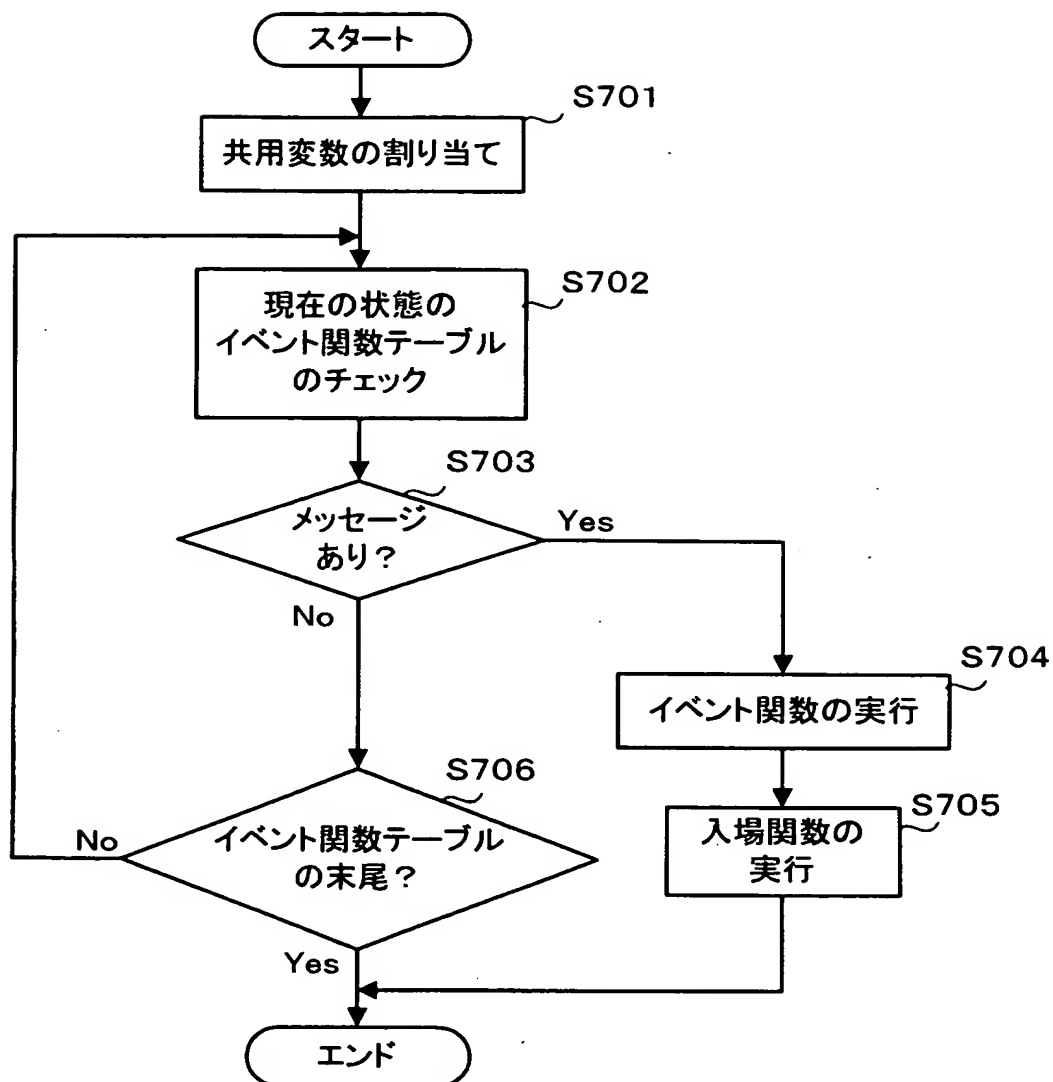
/*****
    evDemoIdle     イベント関数
*****/
void    evDemoIdle(void)
{
    ...
}

/*****
    enterDemoRun  入場関数
*****/
void    enterDemoRun(void)
{
    ...
}

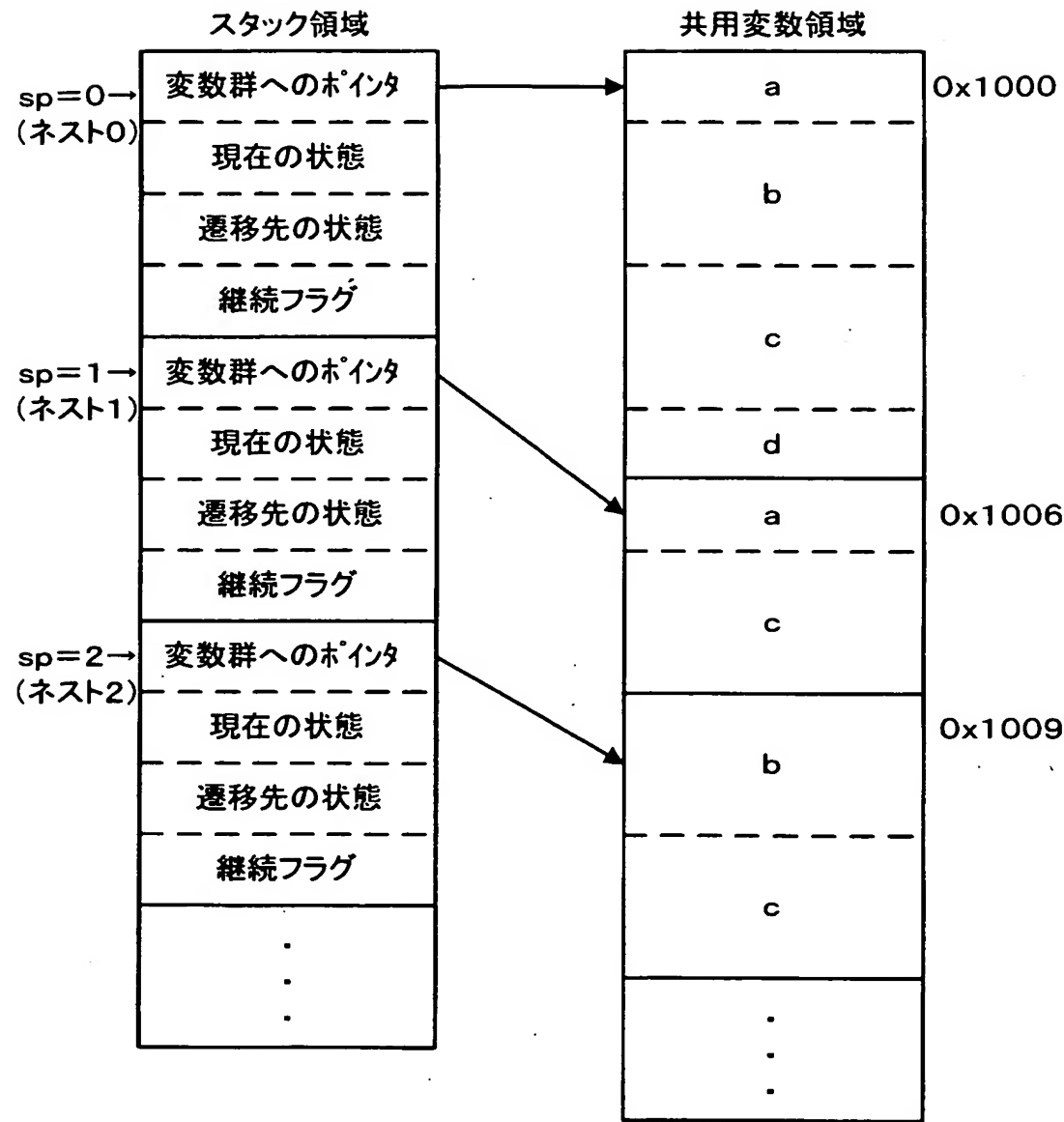
/*****
    evDemoRun     イベント関数
*****/
void    evDemoRun(void)
{
    ...
}

```

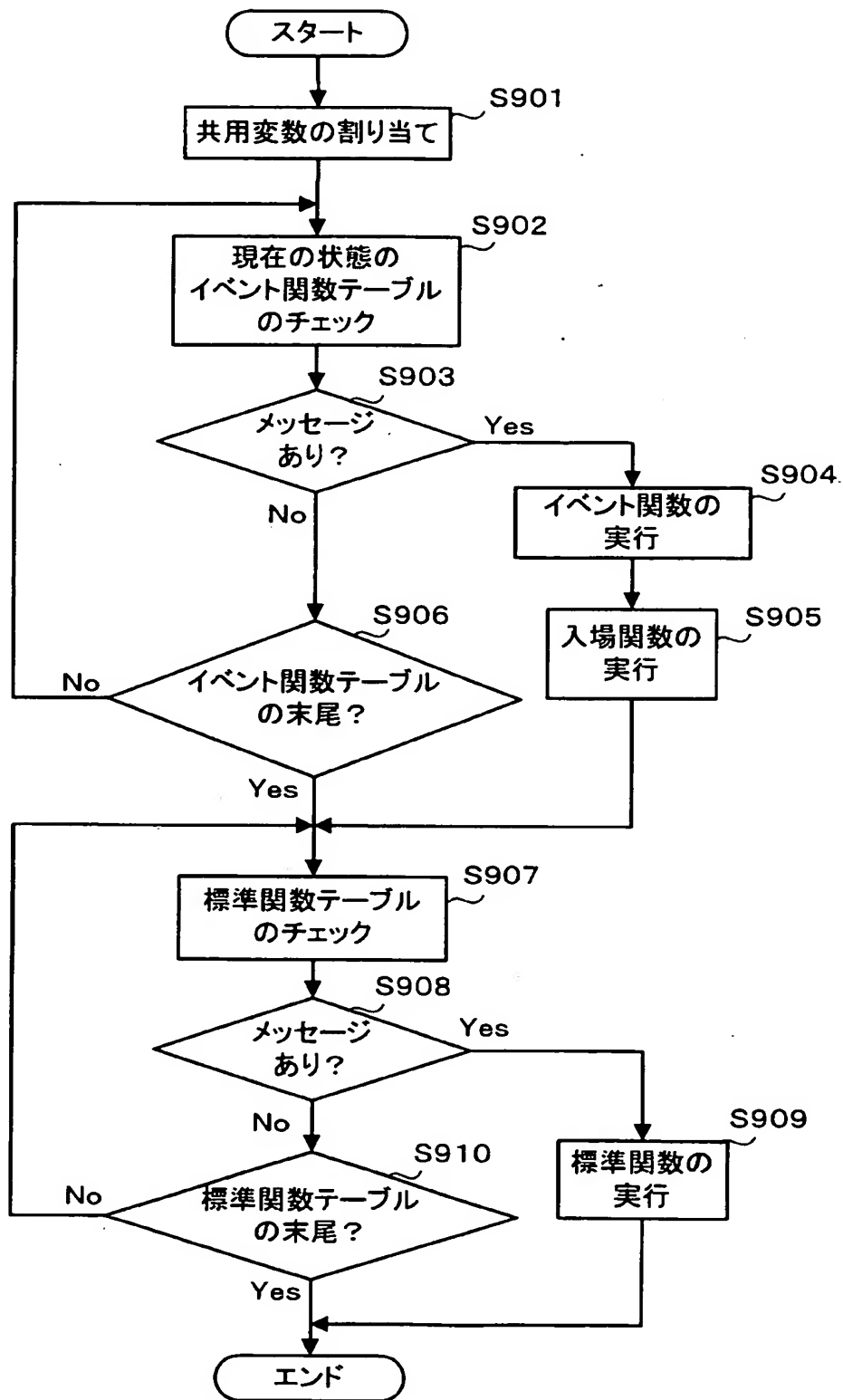
【図 7】



【図 8】



【図 9】



【書類名】 要約書

【要約】

【課題】 画像形成装置の安定性を保持しつつアプリケーションの追加を容易に行うこと。

【解決手段】 画像形成処理にかかるユーザサービスを提供するアプリケーションを備えた画像形成装置であって、外部アプリ 1 1 7 の取り得る状態と、状態下で外部アプリ 1 1 7 に対して生じうるイベントと、イベントが状態下で生じたときに実行されるイベント関数と、イベント関数の実行後の遷移先の状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを含む外部アプリ 1 1 7 に実装され、各モジュールで共有する共用変数をモジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリ 2 1 5 を備えた。

【選択図】 図 2

特願 2 0 0 2 - 2 7 6 5 3 2

出 願 人 履 歴 情 報

識別番号

[0 0 0 0 0 6 7 4 7]

1. 変更年月日 1 9 9 0 年 8 月 2 4 日
 [変更理由] 新規登録
 住 所 東京都大田区中馬込 1 丁目 3 番 6 号
 氏 名 株式会社リコー

2. 変更年月日 2 0 0 2 年 5 月 1 7 日
 [変更理由] 住所変更
 住 所 東京都大田区中馬込 1 丁目 3 番 6 号
 氏 名 株式会社リコー